Versionskontrolle mit Git

Stefan Gast

27. März 2009

Übersicht

Einführung

Geschichte

- Gestartet von Linus Torvalds, dem Initiator und Projektleiter des Linux-Kernels
- Derzeitiger Leiter: Junio Hamano
- Anfänglicher Zweck: Ersetzen von BitKeeper in der Kernelentwicklung (Lizenzprobleme)
- Wird nun von mehreren großen Projekten verwendet, unter anderem:
  - Git
  - QT (Programmbibliothek)
  - VLC (Media-Player)
  - Linux-Kernel
  - GNOME (stellt gerade um)

zum Namen...

„git“ bedeutet auf Englisch soviel wie „Dummkopf“ oder „Depp“. Linus Torvalds dazu:

> *„I'm an egoistical bastard, and I name all my projects after myself. First Linux, now git.“*

Eigenschaften

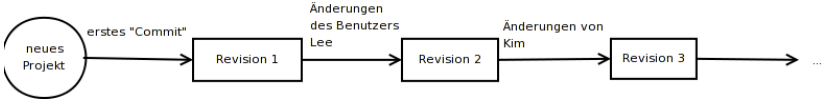
- Schwerpunkt: Nichtlineare Entwicklung
  - Schnelles Anlegen und Verschmelzen von Entwicklungszweigen
  - Werkzeuge zur grafischen Veranschaulichung der Entwicklungsgeschichte
  - Aber auch: Keine Revisionsnummern wie z.B. bei SVN
- Trotzdem wird kein Entwicklungsmodell vorgeschrieben, auch lineares Vorgehen möglich
- Verteilte Entwicklung: Jeder Programmierer hat Zugriff auf die komplette Projektgeschichte
  - Auch ohne Server und ohne dass alle Teammitglieder ständig erreichbar sind

Eigenschaften

- Vielfältige Möglichkeiten zur Veröffentlichung von Repositories, z.B. über: HTTP, FTP oder eigenes Protokoll
- Vollständig kompatibel zu SVN-Repositories (über git-svn)
- Besonders geeignet für sehr große Projekte
- Flexibel
  - Baukasten-Design
  - Erweiterbar durch eigene Skripte

Lineare Entwicklung

- Traditionelle Vorgehensweise: Im Wesentlichen ein Hauptentwicklungszweig, an dem gearbeitet wird

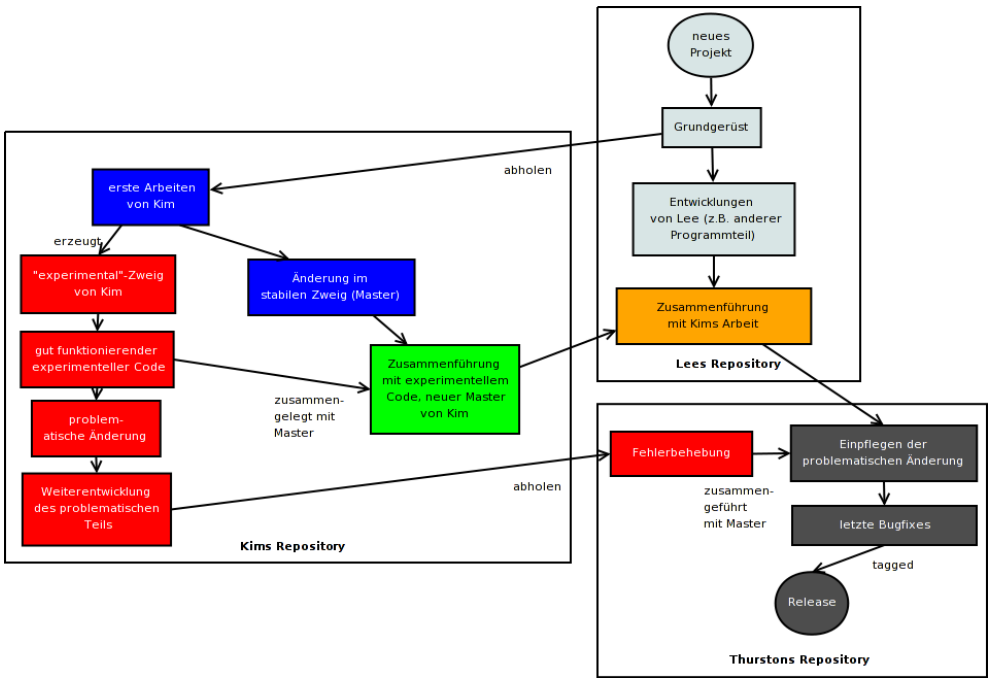


- Nachteile:
  - Wenig Möglichkeiten zum Experimentieren (Hauptzweig darf nicht kaputt gehen)
  - Ineffizient bei sehr großen Projekten (z.B. Linux-Kernel)
- Sinnvoll für kleinere bis mittlere Projekte, da streng geregelter Ablauf

Nichtlineare Entwicklung

- Jeder Benutzer hat seinen eigenen Entwicklungszweig („branch“) und kann bei Bedarf beliebig viele weitere anlegen
- Meistens kein zentrales Repository (verteilte Versionskontrolle)
- Änderungen anderer Entwickler können jederzeit „abgeholt“ werden („pull“)
- Dadurch entstehen sehr viele Zweige, die dann wieder zusammengefügt („merge“) oder auch verworfen werden
- „Cherry-Picking“ möglich, d.h. statt ganzer Zweige können auch gezielt nur einzelne Änderungen übernommen werden

Nichtlineare Entwicklung: Beispiel



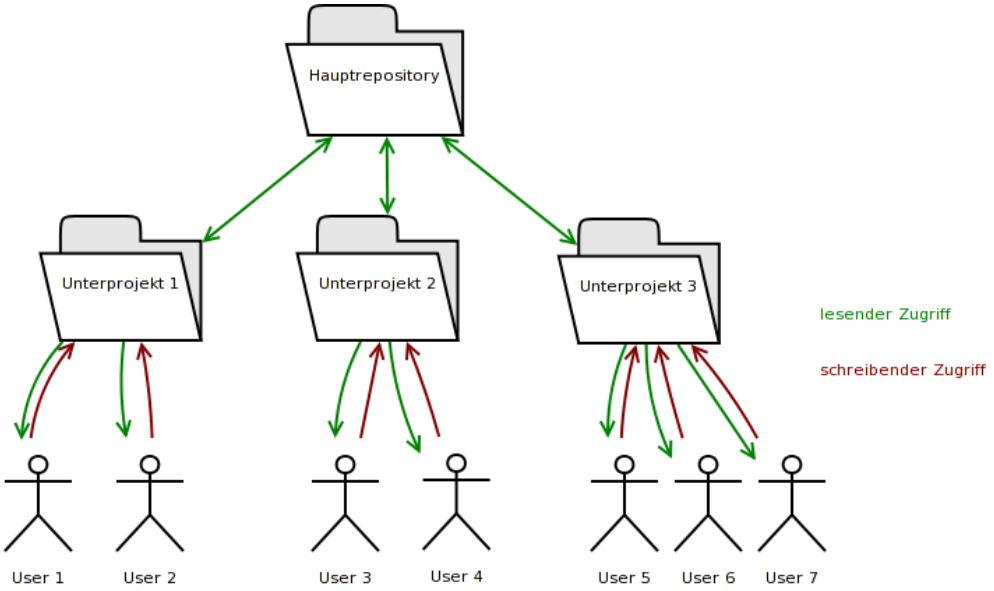
Nichtlineare Entwicklung: Nachteile

- Kann leicht ins Chaos ausarten
- Deshalb: Klare Aufgabenverteilung nötig
- Entwickler müssen wissen, wann und von wem sie den Zweig bekommen, den sie für ihre Arbeit benötigen (teilweise Abhilfe durch Tags)

In der Praxis

- Projekt wird in einzelne Teilprojekte untergliedert
- Ein zentrales Repository, welches sich nach Bedarf oder in regelmäßigen Abständen die Änderungen der Unterprojekte zieht und einpflegt
- Schreibender Zugriff auf das zentrale Repository nur durch den Projektleiter
- Verwaltung der Änderungen in den Unterprojekten häufig wie bei SVN
- Unterprojekte holen zentrale Änderungen bei Bedarf bzw. nach Aufforderung durch Projektleiter aus dem Haupt-Repository

In der Praxis



- Hierarchische Projektstruktur
- Klare Aufgabenverteilung durch Unterprojekte

Bedienung

Syntax

- Kommandozeilenorientiert
- Aufruf über `git kommando --option1 --option2... objekt`
- Online-Hilfe
  - Allgemein, Liste der wichtigsten Kommandos: `git --help`
  - Ausführlicher: `man git`
  - Zu speziellem Befehl: `git kommando --help`

Zugriff auf Änderungen

- Keine Revisionsnummern, sondern Schlüssel und Tags
- „HEAD“ bezeichnet die letzte eingetragene Änderung des aktuellen Zweiges
- Beispiel: `git diff de013ab79760b..HEAD`
  - Zeigt Änderungen seit Commit de013ab79760b...
  - Erste Zeichen des Schlüssels reichen aus, wenn eindeutig

Vereinfachter Zugriff auf neueste Änderungen

- „HEAD^“ vorletzte eingetragene Änderung
- „HEAD^^“: vorvorletzte Änderung
- „HEAD~4“: 4 Einträge zurückliegende Änderung

Ein neues lokales Repository anlegen

```
cd myproject
git init
git add .
git commit
```

- Macht aus dem Verzeichnis „myproject“ ein Git-Repository
- Berechtigungen entsprechen der Dateimaske des Users, d.h. normalerweise können alle Benutzer des lokalen Rechners lesend zugreifen und nur der Eigentümer selbst darf schreiben
- Dritte Zeile stellt bereits vorhandene Dateien unter Versionskontrolle (vgl. SVN)
- Letzte Zeile: Anfänglicher Eintrag in die Versionsverwaltung, der Startpunkt

Dateien hinzufügen

- Einzelne Dateien: `git add crap.c vista.c ...`
- Ganze Verzeichnisse auch möglich:
  `git add verzeichnis1 verzeichnis2 ...`
  - Neu hinzugekommene Dateien müssen aber explizit eingetragen werden
- Nur Quelldateien unter Versionsverwaltung stellen, keine ausführbaren Programm- oder Objektdateien!

Änderungen eintragen

1. `git commit -a`
2. Beschreibung der Änderung in Editor eingeben
   - Erste 50 Zeichen in der ersten Zeile: Kurzbeschreibung
   - Nach Leerzeile: Ausführliche Erklärung
   - Leere Beschreibung bricht Vorgang ab
3. Empfehlung: Eine Eintragung sollte genau eine komplette Änderung beschreiben, nicht mehr, nicht weniger!
   - Gilt insbesondere bei Zusammenarbeit mit mehreren Entwicklern!

Ausgabe des Projektlogs

1. `git log`
2. Beispieleintrag:

```
commit 2255b831f19a31b8aa701f0e08f...
Author: Stefan Gast <mrsteven@gmx.de>
Date:   Wed Mar 18 18:04:02 2009 +0100

Beispiel zu diff
```


Unterschiede zwischen Änderungen anzeigen

- Änderungen seit letztem „commit“: `git diff`
- Vergleich zweier Eintragungen: `git diff schluessel1..schluessel2`
  - Schlüssel entsprechen denen aus dem Log
  - Z.B. Vergleich der zwei vorletzten Versionen:
    `git diff HEAD~2..HEAD~1`

Noch nicht eingetragene Änderungen verwerfen

- Änderungen im Arbeitsverzeichnis, die noch nicht mit `git commit -a` ins Repository eingetragen wurden, können einfach verworfen werden
- Befehl: `git reset --hard HEAD`
- Betroffene Dateien sollten dabei im Editor geschlossen sein

Einzelne Änderung rückgängig machen

- Befehl: `git revert schluessel`
- Falls dabei Konflikte entstehen (gleiche Zeile wurde später erneut geändert) müssen diese von Hand aufgelöst werden

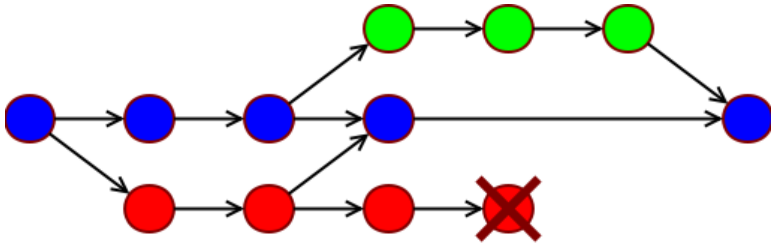
Was sind Tags?

- Kennzeichnungen für bestimmten Entwicklungsstand
  - Beispielsweise: Release, Beta-Version, zum Abholen durch anderen Entwickler bereiter Stand, sonstiger Meilenstein
- Gleichzeitig Kurzschreibweisen für Commit-Schlüssel
  - Beispiel: `git log v1.0..v1.1`

Anlegen, Auflisten und Löschen von Tags

- Anlegen mit `git tag tagname stand`
  - Stand kann wieder Zweigname oder Commit-Schlüssel sein - oder leer für die aktuelle Version des ausgewählten Zweiges
  - Beispiel: `git tag v1.0`
- Liste aller Tags: `git tag`
- Tag löschen: `git tag -d v1.0`
  - Löscht nur das Tag (den Namen), keine Commits!

Zweige allgemein



- Empfohlen wenn sich die Entwicklung an einer bestimmten Stelle verzweigt
  - Testen neuer Ideen
  - Aufteilen der Arbeit auf mehrere Personen
- Zweige („branches“) können zu einem späteren Zeitpunkt wieder zusammengeführt werden („merge“)

Zweige in Git

- Mit Git schnell und einfach
  - *Das* „Killerfeature“ von Git
  - Bei SVN z.B. nur auf dem Server möglich, keine lokalen Zweige
- „master“: Hauptentwicklungszweig des Repositories

Arten von Zweigen

- Lokale Zweige („topic branches“)
- Remote Branches

Lokale Zweige erzeugen

- An aktuellem Entwicklungsstand: `git branch mybranch`
- Von bestimmter Stelle starten: `git branch mybranch startpunkt`
  - Startpunkt kann sein:
    - Anderer Zweig, dann wird der derzeitige Stand dieses Zweiges als Basis verwendet
    - Schlüssel oder Tag eines Commits

Wechseln des aktuellen Zweiges

1. Anzeigen aller Zweige: `git branch`
2. Umschalten auf anderen Zweig: `git checkout otherbranch`
   - Arbeitsverzeichnis sollte sauber sein, d.h. keine noch nicht mit `git commit -a` eingetragenen Änderungen
   - Auch hier wieder: Dateien vorher schließen!

Zusammenführen von Zweigen

1. Zum Zielzweig wechseln: `git checkout targetbranch`
2. Änderungen aus Quellzweig übernehmen: `git merge sourcebranch`
   - Statt Angabe eines Zweiges auch Schlüssel möglich, dann werden nur die Teile bis zu diesem Schlüssel übernommen
3. Falls nötig Konflikte auflösen

Einmal tief durchatmen - Lösen von Konflikten

1. Dateien, in denen Konflikte auftraten von Hand editieren (Stellen sind markiert)
2. Möglichkeiten:
   - Nach dem Editieren konfliktfreie Version übernehmen: `git commit -a`
   - Vorgang abbrechen: `git reset --hard`

Einführung: Öffentliche Repositories

- Nötig, da Rechner der Projektbeteiligten normalerweise nicht immer erreichbar
- Voraussetzung: Ein Server auf den alle Teammitglieder Zugriff haben
- Einfachste Lösung: Server mit SSH, passende Dateirechte für das Repository

Öffentliches Repository erstellen

```
# auf Server
newgrp projektgruppe
mkdir myproject.git
cd myproject.git
git init --bare --shared=group
```

- Erzeugt ein Repository, in das die Unix-Benutzergruppe „projektgruppe“ schreiben kann
  - Zentraler Server
  - Gleiches Vorgehen wie bei SVN möglich
- `--bare` verhindert, dass das Verzeichnis gleichzeitig als Arbeitskopie dient
  - Projektdateien nicht direkt ohne Git sichtbar
  - Sinnvoll bei Repositories, auf die nur von anderen Rechnern zugegriffen wird (braucht weniger Speicher)

Lokale Kopie eines Repositories erstellen

```
git clone ssh://gasts@uranus2.hs-weingarten.de/\
home/PUB/Stud/Stud/gasts/testrepo.git
```

- Kopiert den aktuellen Entwicklungsstand von „...testrepo.git“ auf den lokalen Rechner
- Zweige des Originals sind als Remote Branches unter „origin/xyz“ erreichbar
- Nur „master“-Zweig des Servers wird mit lokalem Zweig verbunden, den Rest muss man explizit lokalen Zweigen zuordnen

Änderungen vom Server in lokalen Zweig übernehmen

```
git checkout master
git pull origin
```

- Holt sich die Änderungen aus dem „master“-Zweig des Servers und versucht diese zu übernehmen
- Bei Konflikten gleiches Vorgehen wie beim Verschmelzen von lokalen Zweigen
- Im Wesentlichen das gleiche wie `svn update` bei Subversion

Änderungen in lokalen Zweigen an den Server übermitteln

- Im Gegensatz zu SVN werden Änderungen bei `git commit -a` nicht automatisch an den Server geschickt!
- Befehl dafür: `git push origin`
- Entsprechung für `svn commit`:
  - `git commit -a && git push origin`
- Falls es hierbei Konflikte gibt muss vorher der lokale Zweig mit `git pull origin` auf den neuesten Stand des Servers gebracht werden
- „commit“ vor „push“ nicht vergessen!
- Empfehlung: Nur in mit `git init --bare` angelegte Verzeichnisse „pushen“

Remote Branches

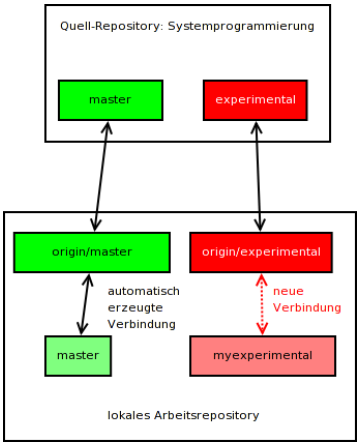
- Benötigt zum Verfolgen von Änderungen in Zweigen auf anderen Rechnern
- Kein direkter Schreibzugriff - erst mit lokalem Zweig zusammenführen
- Beim Klonen von Repositories:
  - Remote Branches für alle Zweige der Quelle
  - Jedoch wird nur der lokale „master“ mit dem der Quelle verknüpft!

Zusätzliche Verknüpfungen zu Remote Branches

- Will man mit weiteren Remote Branches arbeiten, muss man diese mit lokalen Zweigen verbinden
- Neuen lokalen Zweig anlegen, der dem Remote Branch „experimental“ aus der Quelle verbunden ist:

  `git branch myexperimental origin/experimental`
- Danach die übliche Vorgehensweise mit `git pull` und `git push`

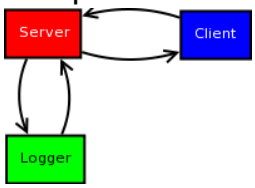
Beispiel




```
git branch myexperimental origin/experimental
git checkout myexperimental
git pull
# editieren
git commit -a
git push
```


Verteilte Entwicklung

- Bisher: Ähnliches Vorgehen wie bei SVN
  - Nur ein öffentliches Repository wird verfolgt (origin)
  - Ein zentraler Server, auf den alle Änderungen geschrieben werden
- Verteilte Entwicklung:
  - Änderungen aus mehreren Repositories verfolgen und einpflegen
  - Meistens nur lesender Zugriff auf fremde Repositories
  - Beispiel:



Mit weiterem Repository verbinden

```
# in Repository server
git remote add logger \
ssh://ranaldol@uranus2.hs-weingarten.de/home/PUB/ \
Stud/Stud/sysprog21/logger.git
git fetch logger
```

- Fügt das angegebene Repository als „logger“ hinzu
- Zweite Zeile erzeugt Remote Branches („logger/xyz“) und bringt diese lokal auf den neuesten Stand

Übernehmen von Änderungen

1. Zu Zielzweig wechseln, falls nötig: `git checkout master`
2. Remote Branches updaten: `git fetch logger`
3. Änderungen ansehen: `git diff HEAD..logger/master`
4. Übernehmen: `git merge logger/master`

Quellen und weitere Informationen

- Allgemein
  - Git Homepage: `http://git-scm.com/`
  - Git User Manual: `http://www.kernel.org/pub/software/scm/git/docs/user-manual.html`
  - Everyday GIT with 20 Commands Or So: `http://www.kernel.org/pub/software/scm/git/docs/everyday.html`
  - `man git`
  - engl. Wikipedia: `http://en.wikipedia.org/wiki/Git_(software)`
- Zweige auf Server anlegen
  - Artikel bei Zorched / One Line Fix: `http://www.zorched.net/2008/04/14/start-a-new-branch-on-your-remote-git-repository/`